Estudos sobre Redes Neurais Convolucionais

Disciplina PSI5886

Prof. Emilio Del Moral Hernandez

Bruno Canale
Bruno Giordano
Fábio Sancinetti
Wanderson Ferreira

December 7, 2016

Objetivos do Trabalho

Os objetivos principais do trabalho foram:

- Entender como as **MLPs** clássicas evoluíram para o que hoje é conhecido como Deep Learning
- Estudar a estrutura e o funcionamento de Redes Neurais Convolucionais 2D
- Entender como a informação é transformada dentro da rede neural ao avançar nas camadas mais profundas
- Extrapolar esse conhecimento para redes mais clássicas como a **MLP** estudada durante a disciplina.

Introdução e Motivação

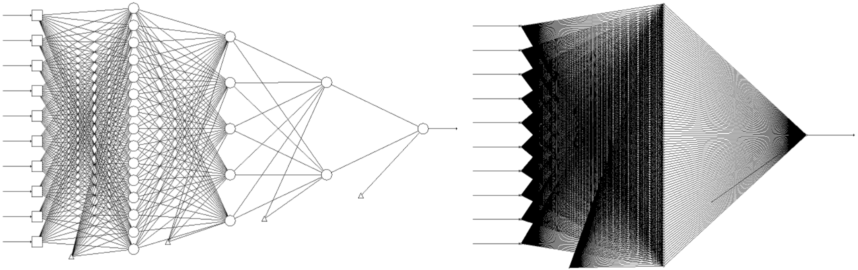
- Cybenko prova que uma rede neural MLP com uma camada escondida e com número arbitrário de neurônios consegue aproximar qualquer função
- Considerando tal resultado, por que tentar fazer redes neurais profundas?
- Delalleau e Bengio fizeram um estudo teórico [1] com neurônios simples (chamados de sum-product units) comparando a quantidade de neurônios necessária para uma aproximação com redes "superficiais" e "profundas"
- O trabalho conclui que para algumas classes de funções a utilização de redes neurais como postuladas por Cybenko requer um número de neurônios consideravelmente maior à uma rede neural com mais camadas escondidas
- Tarefas clássicas de inteligência artificial (como reconhecimento de imagens) se encaixam em algumas dessas premissas, justificando a utilização de arquiteturas profundas nesses problemas

[1] O. Delalleau, Y. Bengio - *Shallow vs Deep Sum-Product Networks*, Neural Information Processing Systems

Esboço da idéia do trabalho

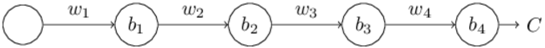
- **Uma rede com n inputs e profundidade $O(\log n)$ pode representar com $O(n)$ units o que uma rede com profundidade 2 representaria com $O(2^{\sqrt{n}})$ units**
- Sejam $\begin{cases} \mathrm{N}_1 \text{ com 128 neurônios e profundidade 7} \\ \mathrm{N}_2 \text{ com } n \text{ neurônios e profundidade 2} \end{cases}$

  Temos $\sqrt{128} = 11.3 \Rightarrow 2^{11.3} = 2540$. Ou seja, N_2 necessitaria de quase 20 vezes mais neurônios para representar uma mesma função f.



Análise preliminar da MLP clássica vista em sala

Para entender os problemas no aprendizado de redes com grande número de camadas escondidas em uma MLP clássica, vamos relembrar alguns conceitos de rede neurais com a arquitetura profunda mais simples possível [2] :



Seja a_i o output do i-ésimo neurônio do modelo. Ou seja:

$$a = \begin{cases} a_1 = \sigma\big(w_1 x + b_1\big) = \sigma(z_1) \\ a_i = \sigma\big(w_i a_{i-1} + b_i\big) = \sigma(z_i) \quad i > 1 \end{cases}$$

Onde $\sigma(.)$ é a função de ativação do i-ésimo neurônio. A derivada parcial de C em relação à w_1 pode ser calculada como:

$$\frac{\partial C}{\partial w_1} = x \cdot \sigma^{'}(z_1) \cdot w_2 \cdot \sigma^{'}(z_2) \cdot w_3 \cdot \sigma^{'}(z_3) \cdot w_4 \cdot \sigma^{'}(z_4) \cdot \frac{\partial C}{\partial a_4}$$

---

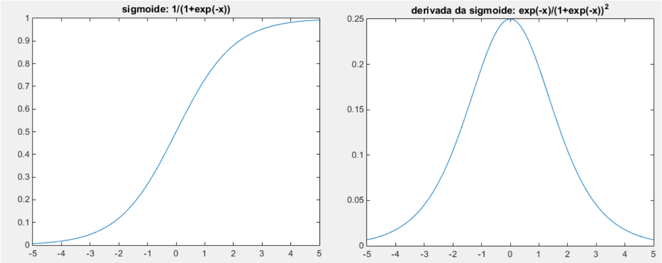
[2] Michael A. Nielsen. - *Neural Networks and Deep Learning*, Determination Press, 2015

Funções de Perda

- Erro quadrático médio (RMS)

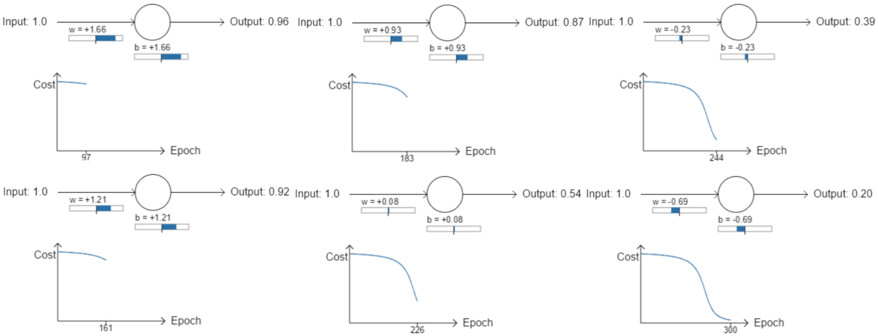
$$C = \frac{(y - a_4)^2}{2} = \frac{(y - \sigma(z_4))^2}{2} = \frac{(y - \sigma(a_3 w_4 + b_4))^2}{2}$$

$$\frac{\partial C}{\partial w_4} = \frac{\partial a_4}{\partial w_4} \frac{\partial C}{\partial a_4} = a_3 \cdot \sigma^{'}(z_4) \cdot (\sigma(z_4) - y_{ref})$$



Funções de Perda

- Treino com RMS



Script disponivel em [3].

[3] Michael A. Nielsen. - *Neural Networks and Deep Learning*, Determination Press, 2015 http://neuralnetworksanddeeplearning.com/chap3.html

Funções de Perda

- Cross-entropy loss:

$$C = -\left(y_{ref} \ln a_4 + (1 - y_{ref}) \ln (1 - a_4)\right) \Rightarrow \left(a_4 = \sigma(z_4)\right) \Rightarrow$$

$$\frac{\partial C}{\partial w_4} = a_3 \cdot \sigma^{'}(z_4) \cdot \left(- \frac{y_{ref}}{\sigma(z_4)} + \frac{(1 - y_{ref})}{1 - \sigma(z_4)} \right) =$$

$$\frac{\partial C}{\partial w_4} = a_3 \frac{\sigma^{'}(z_4)}{\sigma(z_4)(1 - \sigma(z_4))} \left(\sigma(z_4) - y_{ref}\right)$$

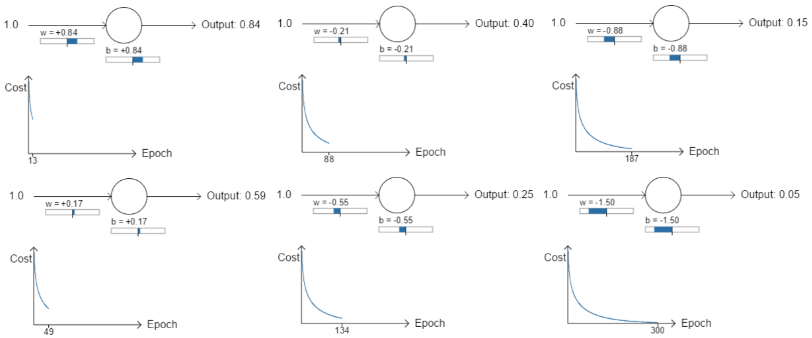
Para a sigmoide $\sigma = \frac{1}{1+e^{-z}}$, temos $\sigma^{'} = \sigma(1 - \sigma)$. Assim:

$$\frac{\partial C}{\partial w_4} = a_3 \frac{\sigma(z_4)(1 - \sigma(z_4))}{\sigma(z_4)(1 - \sigma(z_4))} \left(\sigma(z_4) - y_{ref}\right) = a_3 \left(\sigma(z_4) - y_{ref}\right)$$

- A derivada parcial $\frac{\partial C}{\partial a_4}$ "cancela" o termo $\sigma^{'}(z_4)$.
- Conceito pode ser estendido para outras funções de ativação.

Funções de Perda

- Treino com Cross-Entropy



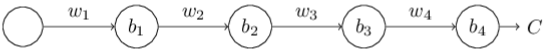
Script disponivel em [4].

[4] Michael A. Nielsen. - *Neural Networks and Deep Learning*, Determination Press, 2015 http://neuralnetworksanddeeplearning.com/chap3.html

Problema Vanishing Gradient

Vamos voltar a analisar $\frac{\partial C}{\partial w_1}$:

$$\frac{\partial C}{\partial w_1} = x \cdot \sigma^{'}(z_1) \cdot w_2 \cdot \sigma^{'}(z_2) \cdot w_3 \cdot \sigma^{'}(z_3) \cdot w_4 \cdot \sigma^{'}(z_4) \cdot \frac{\partial C}{\partial a_4} \rightarrow$$



$$\frac{\partial C}{\partial w_1} = x\left(w_2 w_3 w_4\right) \cdot \left(\sigma^{'}(z_1) \cdot \sigma^{'}(z_2) \cdot \sigma^{'}(z_3) \cdot \sigma^{'}(z_4)\right) \frac{\partial C}{\partial a_4}$$

- Como $\sigma^{'}(z) \leq 0.25$, a multiplicação de vários $\sigma^{'}(z)$ resulta em valores cada vez menores.
- Quanto maior a "distância" entre a camada e a função de perda C, menor a velocidade de aprendizado.
- Se inicializarmos todos os pesos com a mesma distribuição aleatória sem considerar a profundidade da camada em que se encontram, teremos uma inicialização com $\frac{\partial C}{\partial w_1} < \frac{\partial C}{\partial w_2} < \frac{\partial C}{\partial w_3} < \ldots$

Problema Vanishing Gradient

- Rede com 4 camadas escondidas com o mesmo número de neurônios, treinada no dataset MNIST (banco de dados de números escritos à mão)
- Para cada iteração do treino, a norma das alterações Δw_i dos pesos é usada para inferir a velocidade de aprendizado

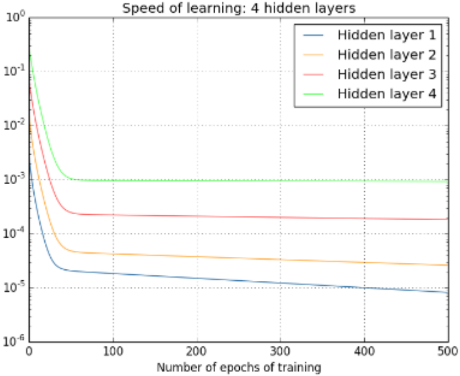


Imagem disponível em [5].

Deep Learning

- Diversos outros problemas associados a escalabilidade de redes neurais
- Estudo de técnicas para possibilitar treinamento de arquiteturas profundas
- Hyper-parâmetros para serem selecionados durante o treino, adequando-o conforme as especifidades dos dados

Regularização

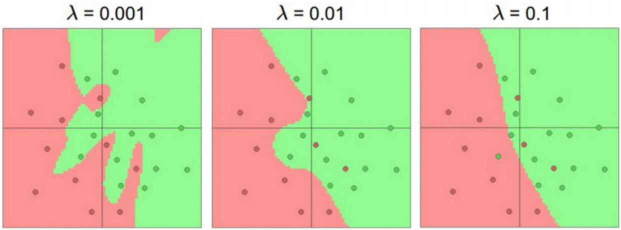
- Consiste na inclusão de um termo extra na função de custo:

$$C = C_{loss} + C_{reg}$$

- Esse termo C_{reg} é usado para penalizar convergências indesejáveis durante o treino, introduzindo novos hyper-parametros
- Regularização L2:

$$C_{reg}^{L2} = \lambda ||w||^2$$

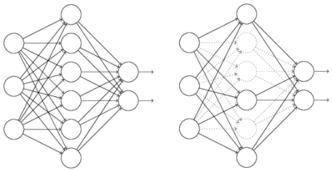
Penaliza otimizações com pesos w altos. Conforme o aumento de λ, as sigmóides tendem a se ativar mais em torno da região linear:



[Andrej Karpathy http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html]

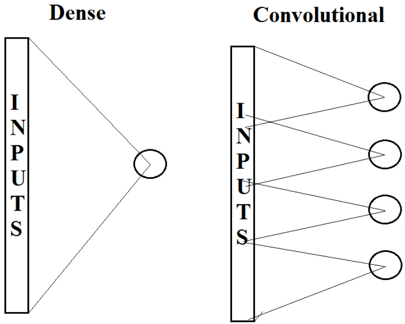
Dropout

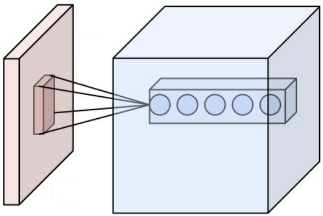
- Técnica para evitar co-adaptação de neurônios
- Em cada iteração de treino, há uma probabilidade p para cada neurônio estar ativo (pesos w mantidos) e (1-p) de estar inativo (pesos w zerados)



- Reduz a co-adaptação de neurônios
- Força os neurônios a aprenderem pesos baseados em diferentes subsets aleatórios de outros neurônios
- Melhora a capacidade de generalização para situações de oclusão de características

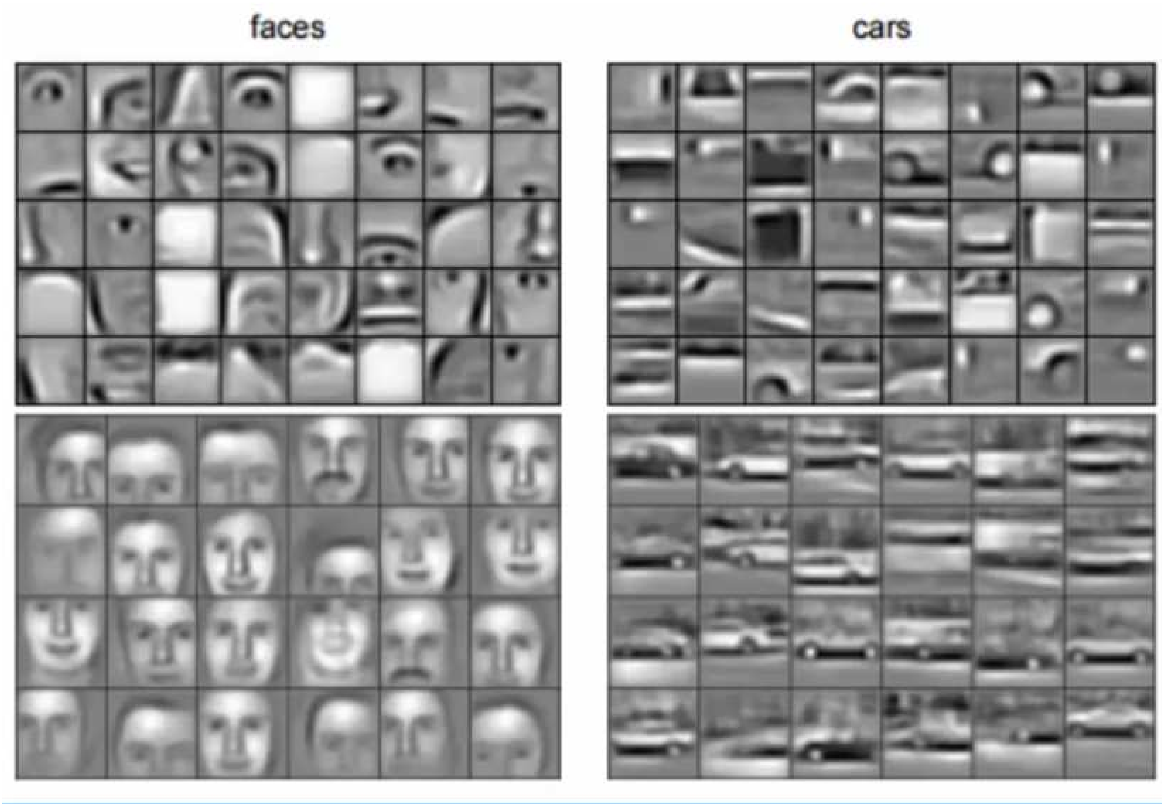
Generalização de arquiteturas de ativação: building blocks





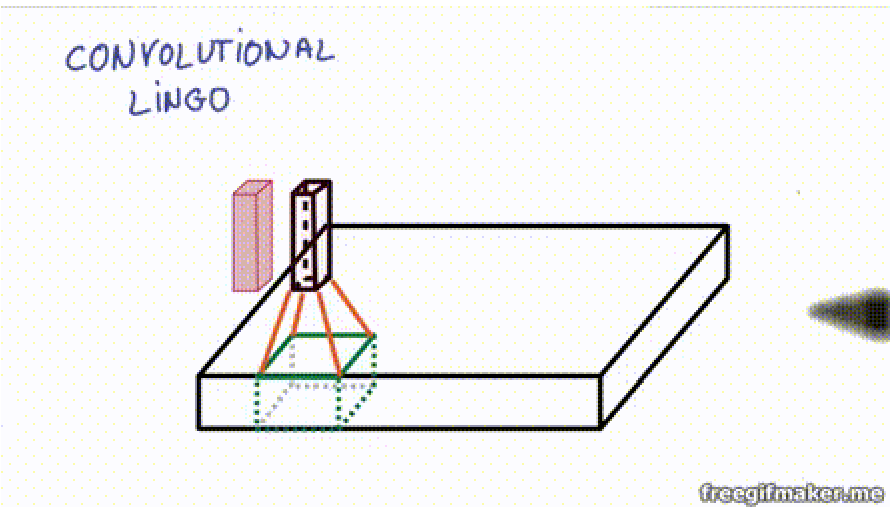
ConvNets - Convolução

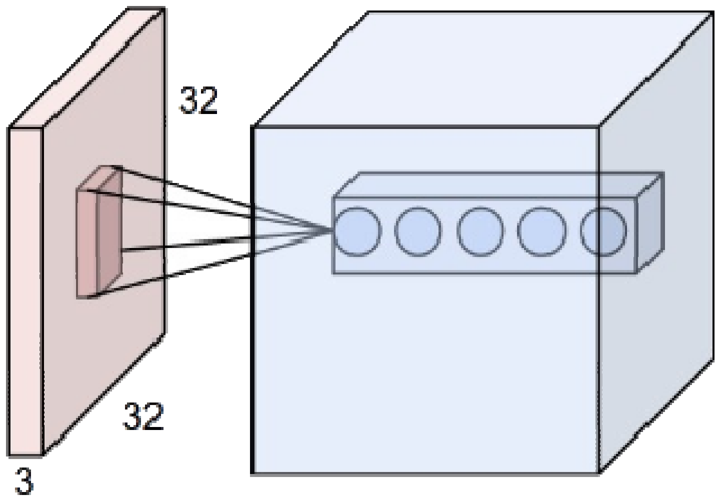
- As redes convolucionais aplicam filtros ao longo da imagem, procurando representações características para então classificá-las



ConvNets – Convolução

- As camadas são volumes que representam convoluções – imagens são filtradas





ConvNets - Convolução

Hiperparâmetros das ConvNets

- **F =** Tamanho do filtro – F x F
- **S =** *Stride* – Deslocamento de *pixels* do filtro na convolução
- **K =** Quantidade de filtros
- ***W*** **=** Tamanho da entrada – W x W
- **P =** *Zero-Padding – Adiciona zeros na periferia das imagens*

Área da Saída!

$$\frac{W - F + 2P}{S} + 1$$

ConvNets – Demais Camadas

Existem outras camadas típicas nas redes convolucionais:

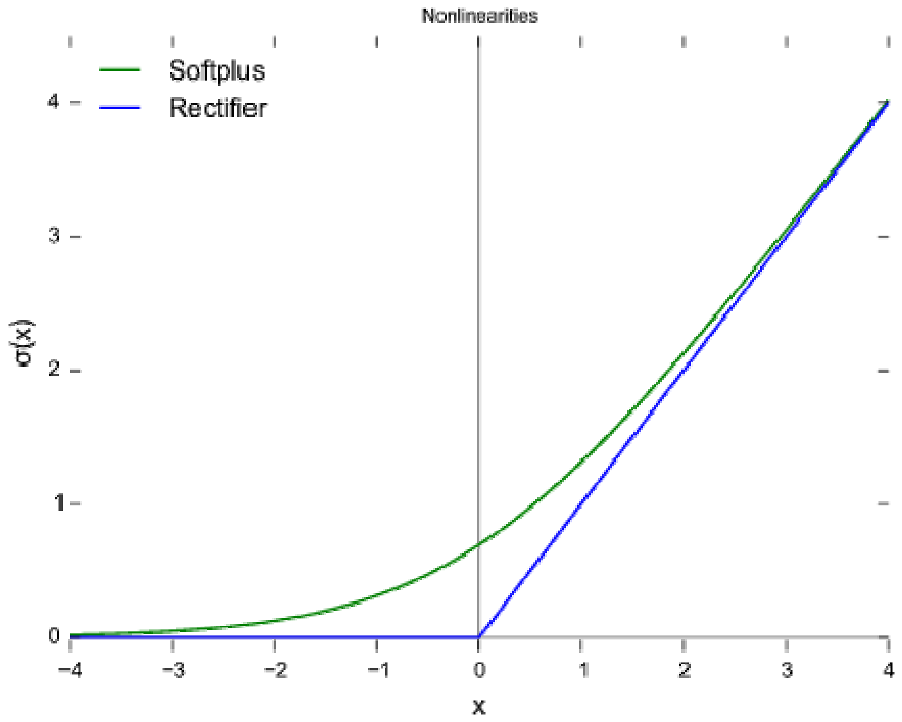
- ReLUs – *Rectified Linear Units*
- *Pooling*
- *Fully Connected – Layer*

ConvNets - ReLUs

Função de Ativação

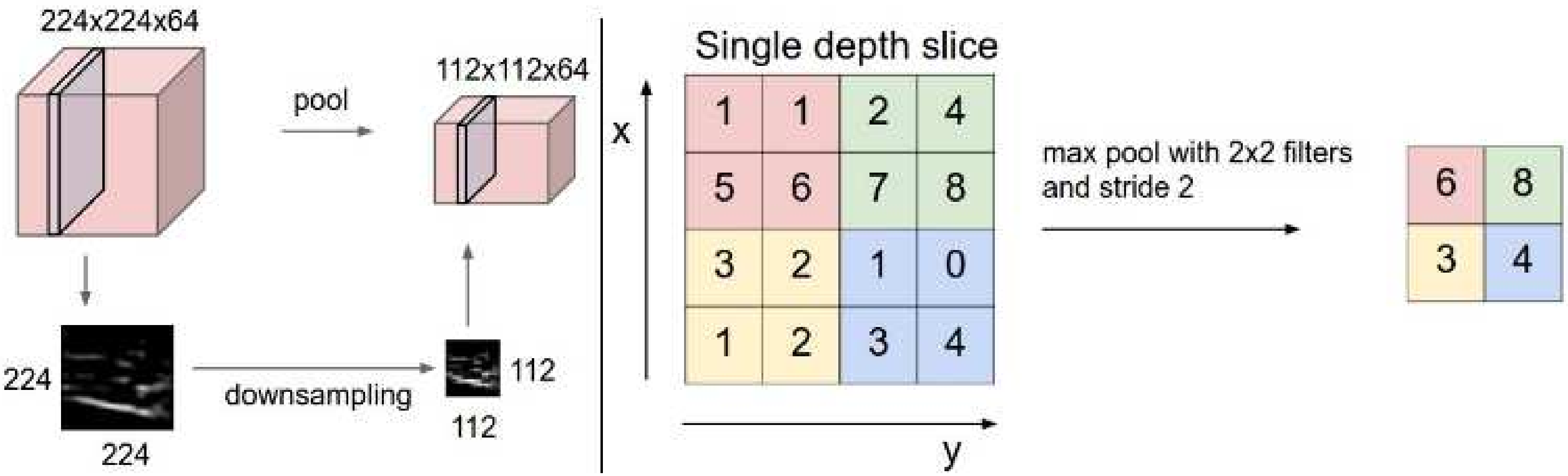
$$f(x) = \max(0, x)$$

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases}$$



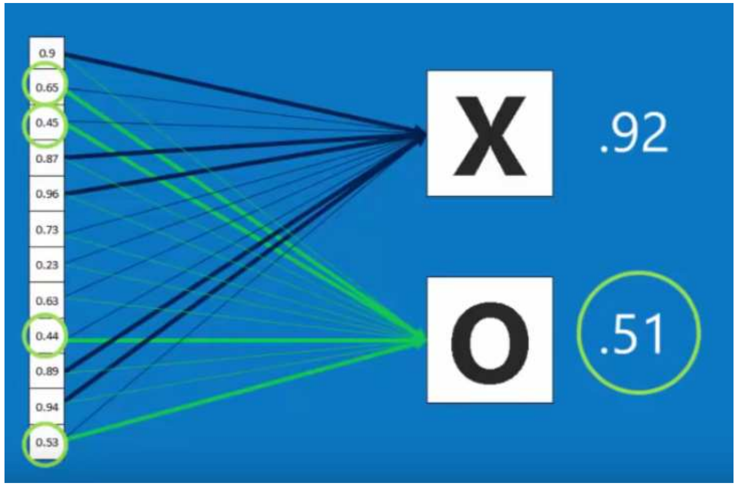
ConvNets – Pooling Layer

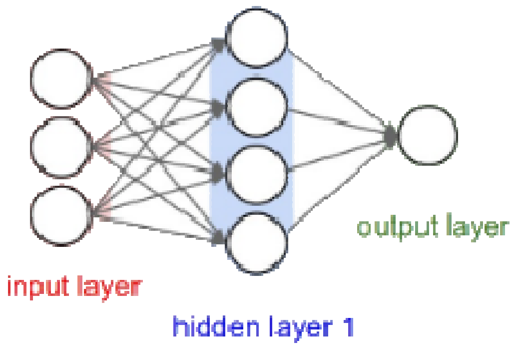
- Camadas com filtros 2 x 2 (F = 2) deslocando dois pixels ao longo da imagem (S = 2)
- De cada janela, extrai-se o maior número
- Realizam *downsampling* nas imagens



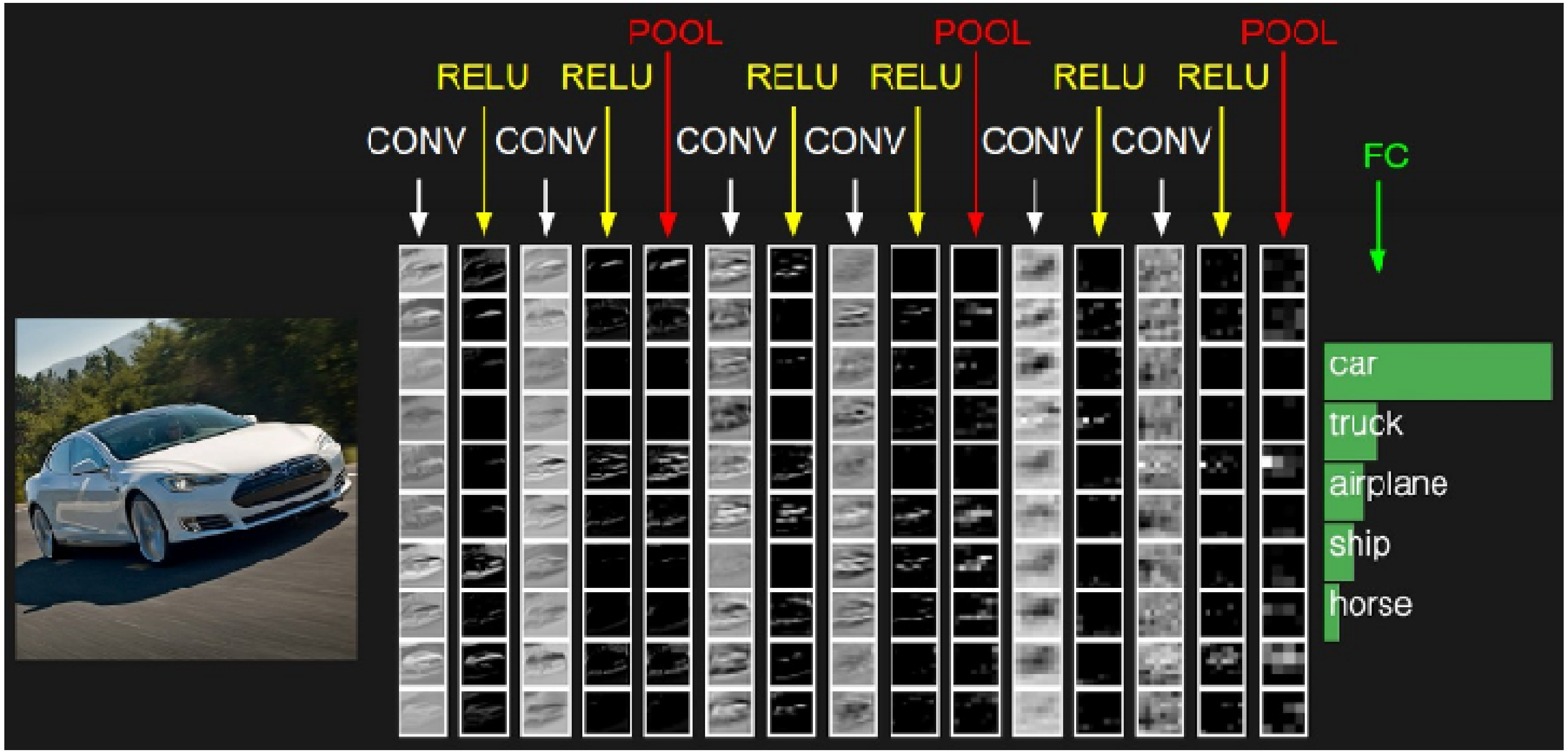
ConvNets – FC Layer

- Com a sequência de combinações de camadas de convolução com *pooling,* a dimensão é reduzida até atingir o formato de um vetor, o qual alimenta uma *fully-connected layer,* permitindo assim a classificação:





Topologia Típica

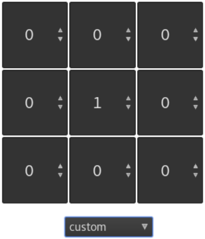


Visualizando a Convolução

`http://setosa.io/ev/image-kernels/`

Explicação visual sobre Convoluções com demonstrações em Javascript

Visualizando a Convolução



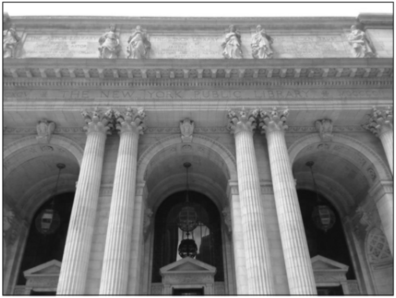


Fig.: Imagem original

Visualizando a Convolução

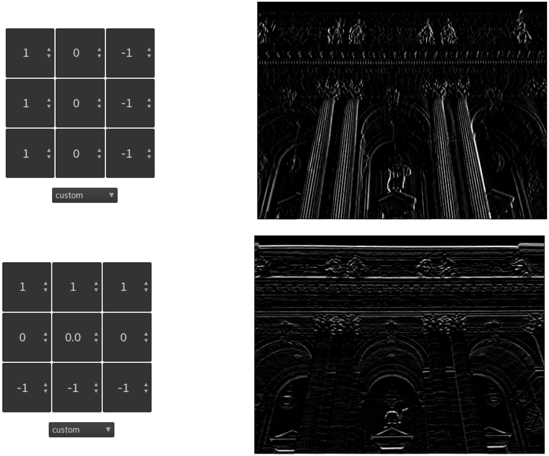


Fig.: Acima: efeito de *Left Sobel*, Abaixo: Efeito *Upper Sobel*

Visualizando a Convolução

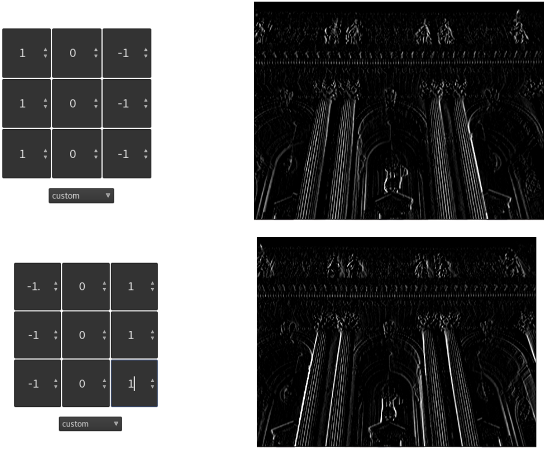
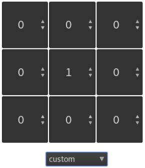
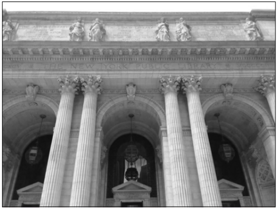


Fig.: Acima: efeito de *Left Sobel*, Abaixo: Efeito *Right Sobel*

Visualizando a Convolução





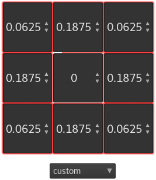


Fig.: Acima: Imagem Original, Abaixo: Efeito *Blur*

Visualizando a Convolução

Fig.: Exemplo trivial de detecção de bordas com filtro convolucional

Base de dados utilizada - MNIST

A base de dados MNIST é composta por 60.000 exemplos de imagens de digitos em letra cursivas. O dataset é ideal para testes de algoritmos em reconhecimentos de padrões por necessitar pouco pré-processamento.
Mais informações no link:
http://yann.lecun.com/exdb/mnist/


```
from keras.datasets import mnist

(X_treino, y_treino), (X_teste, y_teste) = mnist.load_data()
```

Processamento necessário no atributo target

Um processamento padrão é transformar o conjunto de **atributos targets** em um conjunto de variáveis categóricas. O que seriam variáveis categoricas?
Exemplo: Se a lista de targets é composta por: [1.2, 2, 3, 4.2, 4i] e as classes disponiveis são **real, inteiro, imaginario**, então uma matriz de transformação seria:

Table: Conversão para variáveis categoricas

target	real	inteiro	imaginario
1.2	1	0	0
2	0	1	0
3	0	1	0
4.2	1	0	0
4i	0	0	1

Implementação da Rede Convolucional 2D

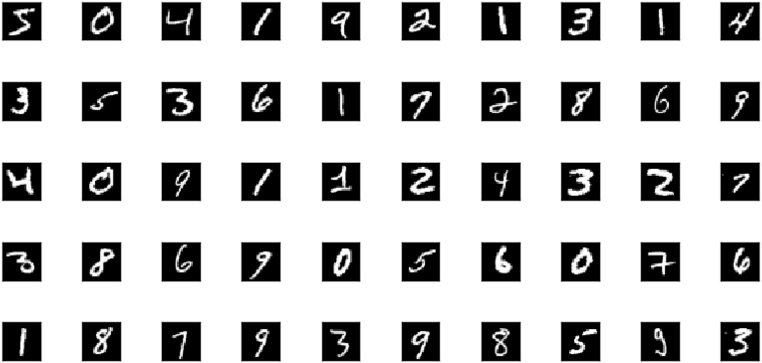
```
from keras.layers import Activation, Dense, Flatten
from keras.layers.convolutional import Convolution2D
from keras.layers.convolutional import MaxPooling2D

num_filtros = 32
num_conv = 6
num_pool = 4

modelo = Sequential() # instaciar o modelo

modelo.add(Convolution2D(num_filtros, num_conv, num_conv,
                         border_mode='valid',
                         input_shape=(28, 28, 1)))
modelo.add(Activation('relu'))

# adicao da segunda camada convolucional
modelo.add(Convolution2D(num_filtros, num_conv, num_conv))
modelo.add(MaxPooling2D(pool_size=(num_pool, num_pool)))

# camada que transforma ('comprime') a saida em um array 1D
modelo.add(Flatten())

modelo.add(Dense(100, activation='relu'))
modelo.add(Dense(numero_classes_categoricas),
           activation='softmax')
```

Modelo do Experimento realizado para análise da Rede

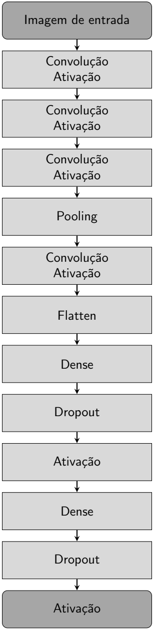
- Após implementação a arquitetura foi testada para verificar performance no conjunto de testes.
- Queriamos observar a transformação da imagem de Input após cada camada da ConvNet
- A fim de analisar com mais detalhes, foram adicionadas mais camadas convolucionais no modelo apresentado anteriormente.

Entradas

MNIST DATASET



Exemplo de Rede



Treinamento

Treinamento

- Épocas $= 10$
- Itens $= 60.000$
- Tempo $= 30 \longrightarrow 40$ minutos

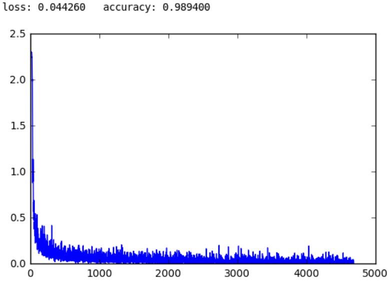
Teste

- Itens $= 10.000$

Treinamento

Resultado na base de treinamento

- 98.94%



Resultado na base de teste

- 99.06%

Convolução - 1

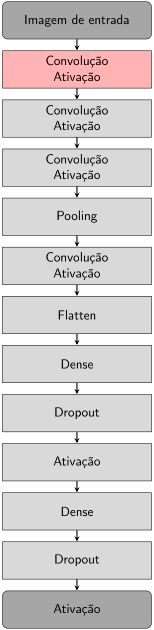


Fig.: Entrada

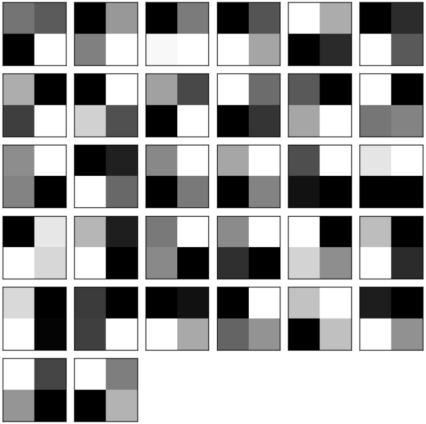
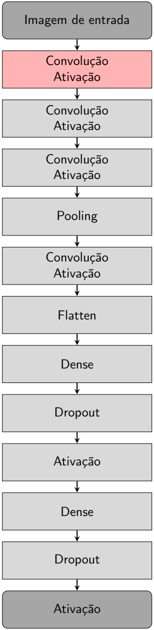


Fig.: Filtros

Convolução - 1



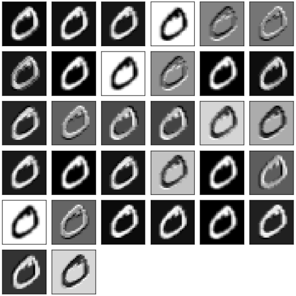


Fig.: Convolução 1

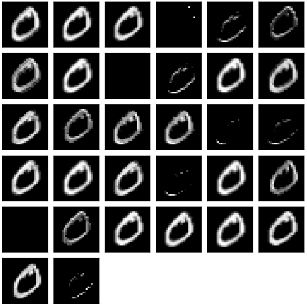
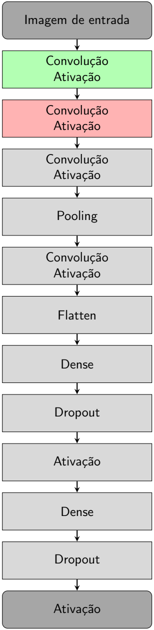


Fig.: Ativação

Convolução - 2



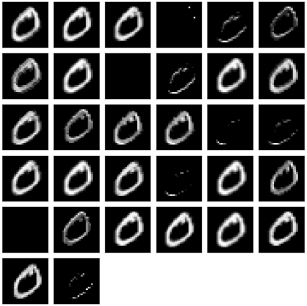


Fig.: Ativação 1

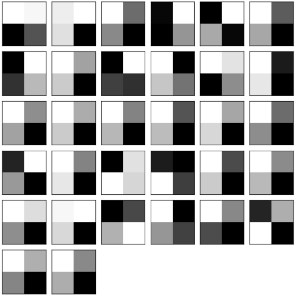
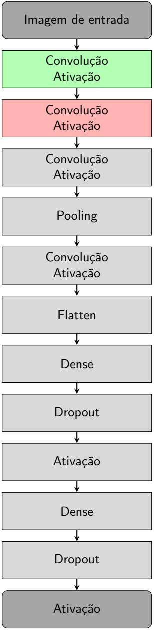


Fig.: Filtros

Convolução - 2



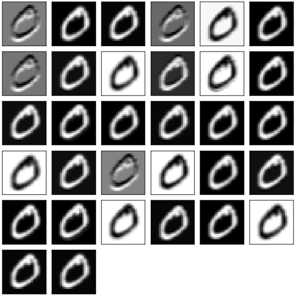


Fig.: Convolução 2

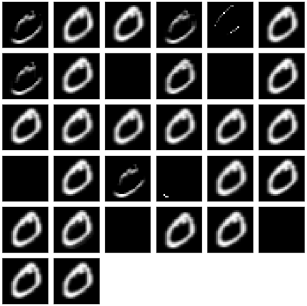
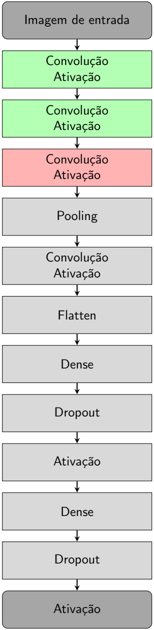


Fig.: Ativação

Convolução - 3



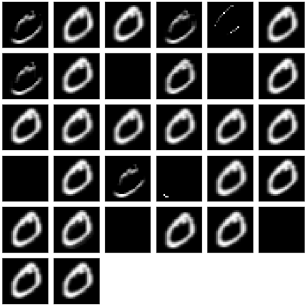


Fig.: Ativação 2

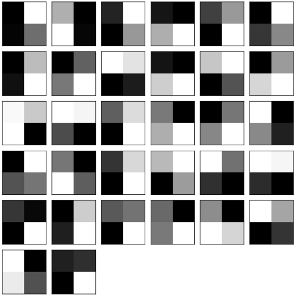
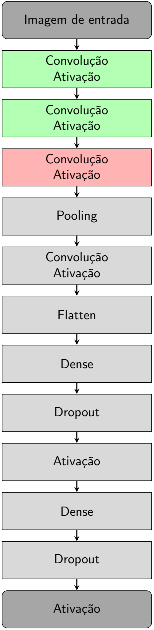


Fig.: Filtros

Convolução - 3



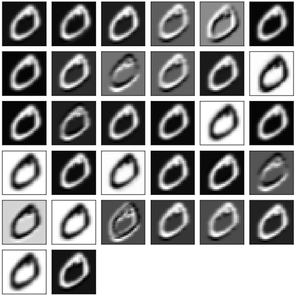


Fig.: Convolução 3

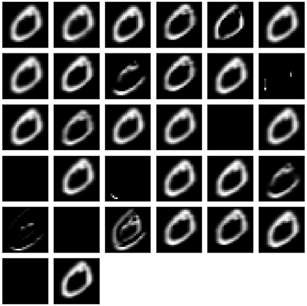
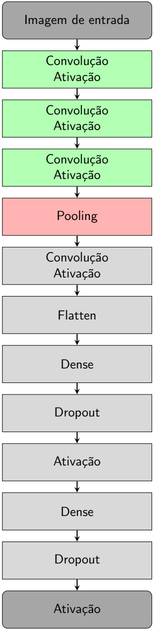


Fig.: Ativação

Pooling - 1



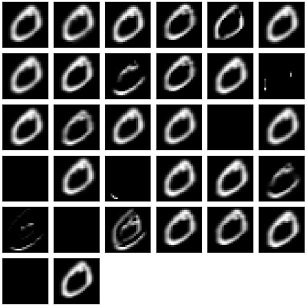


Fig.: Ativação 3

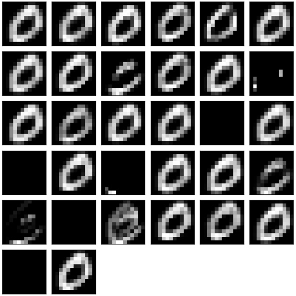
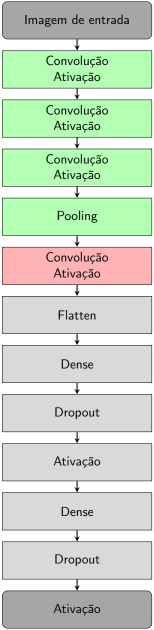


Fig.: Pooling

Convolução - 4



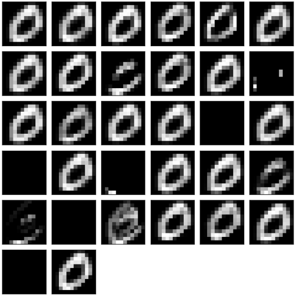


Fig.: Pooling 1

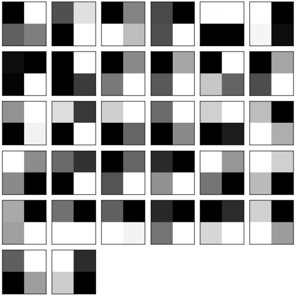
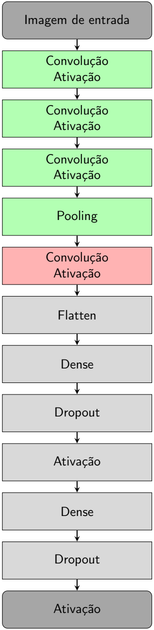


Fig.: Filtros

Convolução - 4



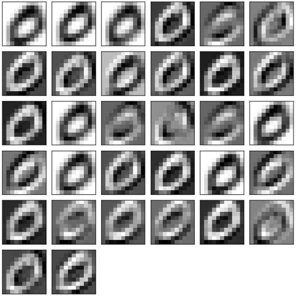


Fig.: Convolução 4

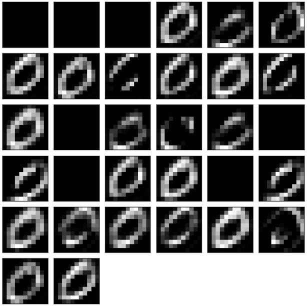
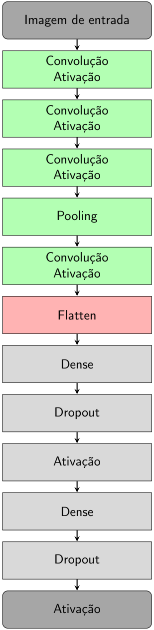


Fig.: Ativação

Flatten



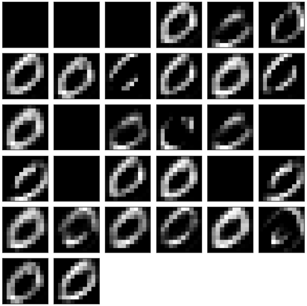


Fig.: Ativação 4

Fig.: Flatten $(3 \times 3 \times 32 \rightarrow 1 \times 1 \times 288)$

Dense - 1

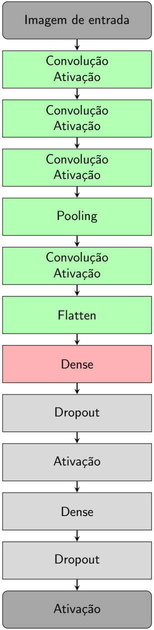
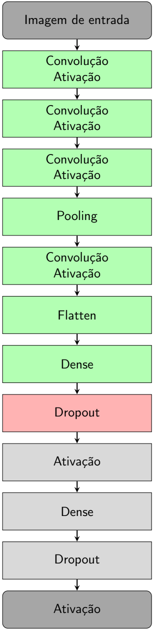


Fig.: Flatten 1x1x288

Fig.: Dense 1x1x100

Dropout - 1



- Dropout $= 0.25$

Ativação - 7

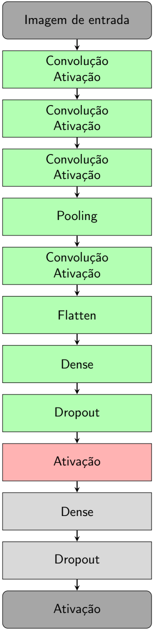


Fig.: Ativação

Dense - 2

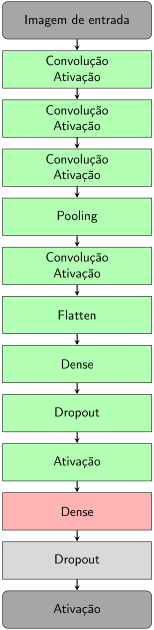
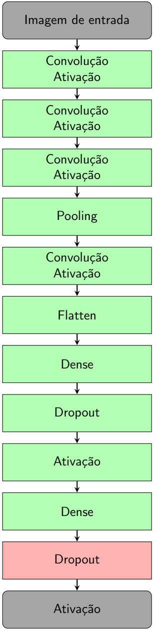


Fig.: Ativação 7 1x1x100

Fig.: Dense 1x1x10

Dropout - 2



- Dropout $= 0.25$

Ativação - 8

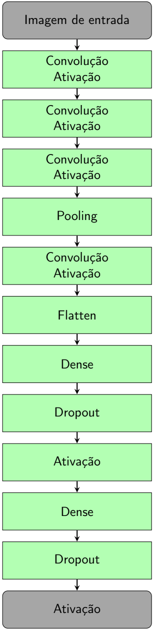


Fig.: Ativação

Outros exemplos de redes convolucionais

- CNNs podem ser utilizadas para entradas bidimensionais como por exemplo reconhecimento de voz.
- *Convolutional Neural Networks for Speech Recognition (Ossama A.H. et Al., 2014 - Microsoft)*

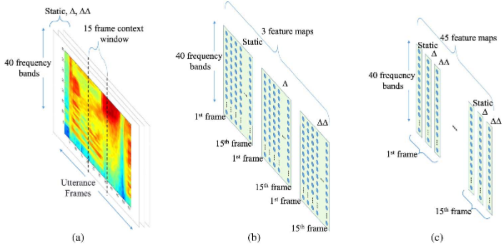


Fig. 1. Two different ways can be used to organize speech input features to a CNN. The above example assumes 40 MFSC features plus first and second derivatives with a context window of 15 frames for each speech frame.

Fig.: Imagem de Ossama A.H. et Al., 2014 exemplificado entrada e processamento no reconhecimento de voz com redes neurais convolucionais

MLP & CNN

- CNN é uma extensão do conceito da MLP
- Convoluções e Pooling ajudam a diminuir rapidamente o número de variáveis do sistema
- Próprio para o processamento de imagens e vídeos